Mapcar: a framework to support the elaboration of the conceptual model of a Knowledge Based System

Pierre Tchounikine

Mapcar: a Framework to Support the Elaboration of the Conceptual Model of a Knowledge Based System

Pierre Tchounikine

IRIN - Université de Nantes & École Centrale de Nantes
2, rue de la Houssinière - BP 92208
44322 Nantes cedex 3
FRANCE

E_Mail: Tchou@irin.univ-nantes.fr

Abstract

Constructing a knowledge based system requires the identification of the conceptual primitives and problem-solving strategy that will permit capturing the expertise in an adapted way. This paper presents an approach to support the elaboration of such a conceptual model by data-abstraction. Our work is motivated by the objective of constructing models that correspond to how experts solve problems, which is required for some applications, as, for example, presenting problem-solving in an educational system. We argue that such an objective cannot be tackled by refining a predefined model. We propose to use a prototype reifying the model as a means for its study. We present methodological guides, and an operationalisation language that allows the putting into practice of the approach we propose. Examples from different applications illustrate the presentation.

Keywords

Conceptual model of a Knowledge Based Systems, modelling by data abstraction, knowledge-level prototyping, operationalisation language.

1. Introduction

Introduced by A.Newell, the ‘knowledge-level’ metaphor expresses the general idea of describing a knowledge based system (KBS) at a level where one can concentrate on the system behaviour and ignore implementation issues [Newell 82]. Such a description is called a ‘knowledge-level model’ or a ‘conceptual model’ (model for short) of the KBS. It can be used for different purposes as “specify the requirements of one or more KBSs”, “drive model-based knowledge acquisition”, “assist evaluation of the correctness and/or completeness of knowledge to improve communication”, “document knowledge in an unambiguous manner, e.g. corporate knowledge assets” [Ai-Watch 95]. Such a conceptual model is generally not something that exists in the head of an expert and can be picked up by knowledge-engineers. It must be constructed by interaction between the domain experts, the knowledge-engineers, and, eventually, specific modules [Linster 93-b].

In this paper we consider the problem of the elaboration of idiosyncratic models, i.e. models that correspond to how experts solve problem. This consists in helping the expert to construct a vocabulary (conceptual primitives) and a problem-solving control that will permit capturing his expertise in an adapted way[1]. Constructing such models is not necessary for a KBS whose only specification is that it should solve problems. It appears as an explicit objective in some other cases, as,

[1]We focus on the model of the task (how problem solving is organised, what is captured by the high-levels of Kads [Wielinga & al. 92] models of expertise) rather than on the model of the domain. By 'conceptual primitives' is therefore intended the primitives used to express how the solving is organised rather than how the domain can be expressed, e.g. what is captured by the epistemological primitives of KL-ONE.

for example, when modelling teachers' problem-solving in an educational software. Such models are not necessarily generic (i.e. reusable for different domains) nor even as rational as could be. Particular aspects of the problem-solving that could be "rationalised" must be respected when they correspond to constraints such as pedagogical necessities [Tchounikine 90].

Constructing models that can be very specific and non reusable is a change in the mission that consider modelling methodologies. Most of these (e.g. Kads [Wielinga & al. 92], GT [Chandrasekaran 87], and Components of expertise [Steels 90]) aim at constructing generic models. For this purpose, they advocate a top-down definition of the model by adapting generic problem-solving methods to the considered domain: knowledge engineering is viewed as an interpretative process. Not underestimating the advantages of this approach to solve problems, we think that matching the observed expertise with predefined patterns of classical knowledge-use is not adapted to the elaboration of idiosyncratic models. We therefore propose to use the alternative bottom-up approach of modelling by data-abstraction from an observed expertise, and to consider knowledge engineering as a constructive process.

The process we suggest (that can be called, following [Karbach & al. 93], 'Knowledge-Level Prototyping', KLP for short) consists in conducting the elaboration of the model in parallel with the construction of a prototype that reifies it and serves as a basis for its study. The prototype is operationalised with a specific high-level operationalisation language, which permits representing the model at an abstract level, making it operational, and analysing it with reflective modules. 'Knowledge-level' denotes that although operational, the prototype contains an abstract description of the model; this allows the study of the model through the study of the prototype. Different other works have emphasised that operationalising the conceptual model helps in the refinement and the validation of the model: Model-K [Karbach & al. 93], Omos [Linster 93-a], [Vanwelkenhuysen & al. 90]. What these works (in particular Linster with the Omos framework) consider is the refinement of generic models. We propose to use such an approach from the first steps of the modelling (Cf. Figure 1).

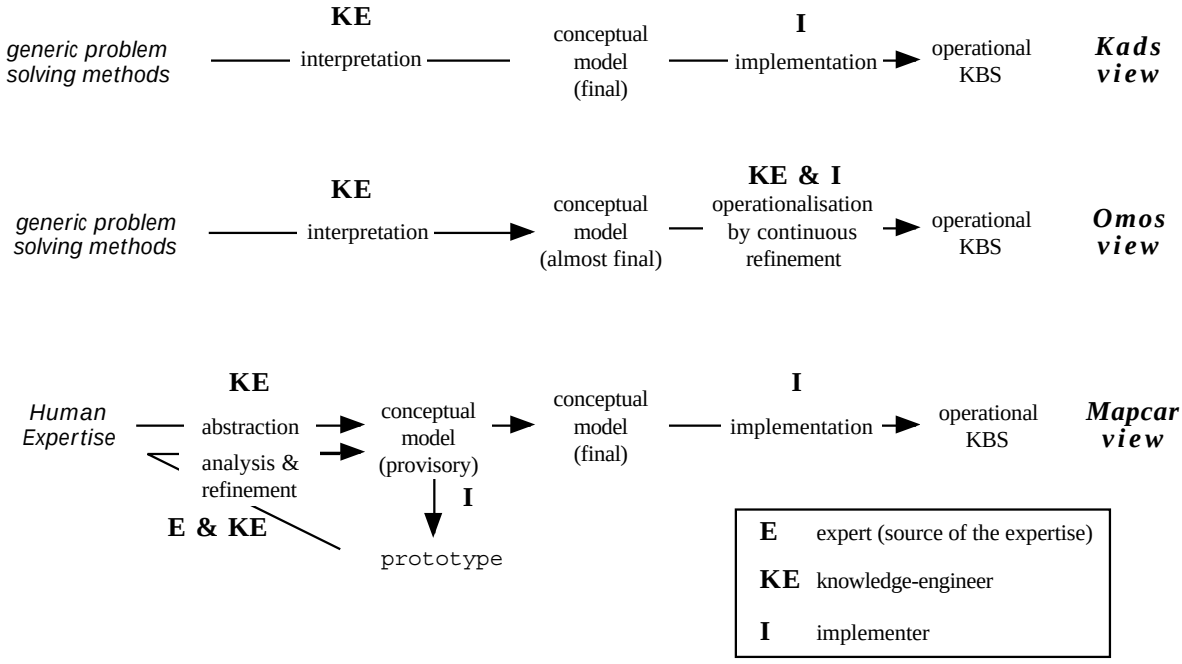


Figure 1. *Mapcar view on modelling.*

KLP as advocated here requires a specific operationalisation language. Different operationalisation languages defined according to a particular type of conceptual model have been proposed. For instance many languages (e.g. Omos and Model-K, see [Fensel & al.94]) directly propose the conceptual primitives of the Kads methodology. This allows a direct mapping paper-based model / operationalisation language, and limits what has to be coded when implementing the prototype. Such languages can be considered as modelling languages, as they impose conceptual primitives and a particular structure of the model (Kads four layers: strategy, task, inference, domain). Some other languages, although not defined according to a methodology, propose predefined structures such as tasks and methods (for instance MML

[Guerrero-Rojo 95] or Lisa [Jacob-Delouis & al. 95]). These structures also impose a modelling point of view. In our approach, the conceptual primitives and the structure of the model cannot be anticipated, and, therefore, a modelling language cannot be used. This led to the design of Zola, an operationalisation language adapted to our objectives [Istenes & al. 96-a].

Zola's main features are (1) capacity to define and make operational specific conceptual primitives, (2) capacity to construct different structures of models, (3) capacity to implement a prototype that reifies the model, and (4) capacity to construct reflexive analysis tools that can inspect the prototype (i.e. the model under construction). Zola allows prototyping a conceptual model. In order to allow reusing as much code as possible while not constraining the modelling by predefined structures, we suggest two approaches:

• elaboration of the model from scratch. This is necessary when the structure of the model is completely specific to the expertise. The help provided in this case is that of the use of Zola, i.e. the capacity to define the model conceptual primitives without low-level implementation problems, the capacity to run a prototype that respects a structural correspondence with the model, prototyping facilities (easy modification of any part of the model), the capacity to construct reflective modules that can analyse the prototype and help in the modelling process.

• reuse of a type of model. A type of model is a possible way to structure a conceptual model. An example is: tasks, methods and a mechanism to dynamically associate methods to tasks[2] . An operational kernel can be associated to such a type of model, and adapted to the specificities of the particular model to be implemented. This allows the minimalisation of the operationalisation by refining predefined pieces of code. It also allows the definition of reflective modules specific to the type of model. Note that a type of model should not be confused with a generic problem-solving method, i.e. a particular way to solve some problems (for example a diagnostic method).

A certain number of works have been undertaken or are ongoing following this approach. Examples of how it allows the definition of a conceptual model are presented in [Tchounikine & al. 95] for a fault diagnosis expert system, in [Tchounikine 94] for an educational software. How the Zola language allows operationalising such models is presented in [Istenes & al. 96-a]. An example of a type of model that performs dynamic selection of tasks and methods is presented in [Istenes & al. 96-b]. In this paper we summarise the motivations, the principles and the putting into practice of our approach.

The paper is organised as follows. In Section 2 we emphasise that the elaboration of idiosyncratic models should be considered as an explicit objective that deserves a specific approach, we argue in favour of tackling this objective by modelling by data-abstraction, and we advocate knowledge-level prototyping to support this process. In Section 3 we describe the approach (termed Mapcar) we propose. In Section 4 we describe the putting into practice of this approach. The main characteristics of the Zola operationalisation language are presented, followed by examples of how this language allows the undertaking of the key points of our approach. Finally, we put some aspects of this work into perspective in Section 5.

It should be noted that what is proposed is a framework to support a constructive view of modelling. There is no claim that all problems of such a modelling are solved. In particular, an essential danger of this approach is to fail in the abstraction and to define a descriptive model of the expert behaviour. This is the main argument in favour of top-down refinement of generic structures. We acknowledge this argument (although it stands less in the context of education, teachers being more able to elaborate a satisfactory model than domain experts). Modelling by data-abstraction is nevertheless required in certain cases, and must therefore be considered. Note that elaborating a model from an observed expertise makes apparent different types of problems such as cognitive problems (e.g. adequacy of the model with the observed performances) or elicitation problems (e.g. reliability of the expert and consensus between experts) [Elliot & al. 95, Gaines & al. 93] that we do not consider in this paper.

2 Within such a structure some flexibility remains, as for instance the precise definition of what is a task or a method (what slots should be defined in order to represent them), which can be adapted to the expertise to be modelled.

2. Elaboration of idiosyncratic conceptual models and knowledge-level prototyping

2.1. Why to elaborate idiosyncratic models

For most KBS', there is no need that the conceptual model should correspond to the human expert problem-solving. Therefore, the fact that the knowledge engineering process influences the obtained model is not critical [Gaines & al. 93]. A usual constraint for the conceptual model is just that it should be understandable by the experts, in order to serve as a guide for the domain knowledge elicitation, or for the understandability of the system's explanations. This is the situation that most knowledge acquisition methodologies consider (e.g. Kads, GT, and Components of expertise), for which reusing or refining predefined generic problem-solving methods is generally considered as an adequate approach. The more rational the model is the better it is, and human reasoning is nothing else than a source of inspiration [Breuker 93].

The situation we want to tackle is when the conceptual model to elaborate must correspond to that of the human experts involved in the construction of the system. In order to understand our objective unambiguously it should be clearly noted that, following Norman's definitions, what we are concerned with is the definition of a *conceptual model* ("...which the expert imputes as underlying his or her behaviour", "that is concerned with communicating with other entities"), and not of the *mental model* of the experts [Gaines & al.93]. In other terms, we don't intend to capture the effective model of the expert, but to construct the model the expert would like to explicit and transmit.

The prototypical case from which our work originates is that of the elaboration of the conceptual model of a knowledge based system that presents problem-solving in an educational system. All authors agree that although problem-solving must be presented in the context of concrete problems, the underlying conceptual model must be presented at an abstract level [Anderson & al. 90, Frederiksen & al. 93, Kieras 88, Means & al. 88, Schaafstal & al. 93, Winkels & al. 92]. Models defined to solve problems cannot generally be used, nor even be modified, for educational purposes [Clancey 87, Wenger 87]. When didactic or pedagogical studies exist, they can serve as a basis for the construction of the model. The model is, nevertheless, often elaborated from the teachers' experience, in particular for ill-studied domains. The objective is to make the system solve problems as teachers do when they interact with their students. The modelling is constrained by other aspects than defining a rational problem-solving model, as for instance dealing with students' knowledge and problem-solving model.

Other contexts require modelling with the constraint of respecting human experts' problem-solving, as for instance when the model must be shared by the KBS and human experts[3]. An example is reported in [Tchounikine & al. 95] in the context of the construction of a fault diagnostic expert system for a company constructing robots. An explicit constraint was that the conceptual model of the system had to be shared by the company's own technicians, the customers' technicians and the company's trainers, to allow communication between the different persons that have to cooperate during a diagnosis. Instead of constructing a system based on a model different from that of the human experts and imposing this model as the common basis for everybody, the company preferred the definition of an 'original company model' that would not change the usual diagnosis activities of the company.

2.2. Abstracting the model

Different views of the elaboration of the conceptual model of a KBS have been proposed. The transfer view suggests that the model is directly accessible. The interpretation view (that adopts methodologies such as Kads[4] or GT) suggests that generic patterns of knowledge-use can serve to interpret the problem-solving to model; the obtained model is a refinement of the identified pattern. The constructive view suggests that the model must be constructed. Elliot & al., in a context of constructing a problem-solving system, propose to adopt one of these views according to a domain fluidity metaphor. When knowledge is "solid" the psychodynamic process is "knowing" and a transfer process is possible; when knowledge is "fluid" the psychodynamic process is "understanding" and an interpretation process is possible; when knowledge is "gaseous" the psychodynamic process is "synthesising" and a constructive process is necessary [Elliot &

[3] For more convenience we will use 'teacher' when we mainly argue in an educational context, and 'expert' when the argumentation is more general.

[4] Note that the Kads project has evolved towards a more constructive view of modelling with CommonKads [Valente & al. 93].

al. 95]. For our objective, the criterion is not the domain knowledge fluidity, but the understanding that one has of the model underlying the target behaviour. If it is directly available the transfer is possible. In general, experts are not able to propose a conceptual model that reflects their competence. An interpretative process can help in structuring the expertise by proposing predefined models, which will be refined according to the considered specificities of the expertise to be modelled. This, nevertheless, necessarily introduces a bias in the modelling: it corresponds either to the definition of a model different from that of the expert or to the projection of the expert expertise on one of the models provided by the considered libraries[5]. In order to respect the experts' problem-solving, a constructive process from his expertise is necessary[6].

In order to capture the expert model, the expert's role cannot be limited to the presentation of 'think aloud' problem-solving. The expert is the source of the model to elaborate and the one that must validate the model. Therefore, he must play an introspective role to help the knowledge-engineer in the modelling, and a controller role during the modelling. This is not an unreasonable request, in particular in the context of education [Gaines & al. 93]. Though they are not knowledge-engineers and cannot be asked to directly propose a model, teachers' capacities make them good assistants in the modelling process. Although pointing to different objectives, the AEM methodology proposed by K.Sandahl is based on a similar attempt to ours and demonstrates that it permits effective results (in the AEM methodology, the expert acts as a "teacher of knowledge structuring" [Sandahl 94]). This makes us diverge from some other bottom-up methodologies such as Macao, where the expert plays no role during the modelling [Aussenac 94].

Sandahl notes that a current misunderstanding of the AEM methodology is to assimilate the expert and the knowledge-engineer. In AEM as in Mapcar, although the expert plays an active role in the modelling, the knowledge-engineer is the one responsible for the modelling: it is not possible to act satisfactorily both as an expert and a knowledge-engineer [Sandahl 94].

2.3. Using a knowledge-level prototype to reify the conceptual model

Prototyping is a software engineering method that aims at tackling a certain number of problems with classical system development, from which, what the user really wants is often only known when the software is finished. The construction of a prototype early in the development cycle helps in solving this problem, by providing a concrete basis for the different persons concerned with the project (users, developers) to understand each other [Budde & al. 91].

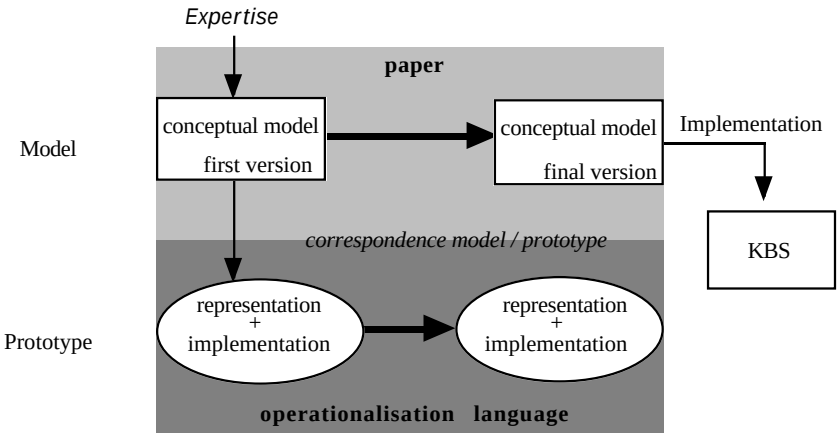
In our context, the user's role is played by the expert whose competence is to be modelled, and the prototype is a means for the expert and the knowledge-engineer to fix the conceptual model. Differently from methodologies where the experts are not supposed to be confronted with the model itself, here the expert is supposed to help in the refinement and the validation of the model in elaboration. Presenting a Kads-like abstract inference structure is not sufficient to give a domain expert (who is not a knowledge-engineer) a clear understanding of a complex model. Sandahl notes that not one of the experts involved in the construction of KBS with the AEM methodology could have been confronted with Kads models of expertise [Sandahl 94]. Experts' usual activity is to tackle concrete problems. A prototype that puts into evidence in concrete terms what problem-solving corresponds to the model is a more valuable way to allow effective critique of the model, and discover new or underestimated modelling problems.

The prototype we need must reify the model under construction. For this purpose, it must respect the structural correspondence principle [Reinders & al. 91], i.e. the fact that every component of the model has a correspondence in the implementation. This allows the refining of the model by studying the prototype (and, vice-versa, facilitates the evolution of the prototype when the model is modified). To avoid low-level implementation details, the prototype must be developed using a high-level operationalisation language that allows a rapid and straightforward representation of the model. Emphasis is on modelling, not on implementation details. Note that the prototype is not the first version of the final software. Figure 2 illustrates this process.

---

[5] Note that, as argued by promoters of the interpretative point of view, a systematic model such as the ones proposed by Kads library can be pedagogically more suitable than the one the teacher can propose [Winkels & al. 92]. This can be true for particular domains but cannot be considered as a general rule. Domain and pedagogical specificities must be taken into account in educational software. Problem solving is presented according to different considerations, e.g. what is the student's knowledge and the interaction context. The more rational model from the problem solving point of view is not necessarily the more pedagogically suitable.

[6] which is not in contradiction with using predefined models in order to refine *in a second phase* the model, as suggested in [Aussenac 94].



The first paper-based satisfactory version of the conceptual model is represented and implemented with a high-level operationalisation language. The constructed prototype serves as a basis for the refinement of the paper-based model. Paper-based model and prototype evolve in parallel. When the conceptual model is fixed it can be implemented with an adapted language.

Figure 2. *Modelling by data-abstraction and knowledge-level prototyping*.

During the modelling process, the prototype represents a partial and/or provisory understanding of the domain expertise. As the model evolves, one can have some difficulty in controlling what the system can and cannot do, or what will happen if one changes this or that in the model. Assistance in seeing what implies changing an aspect of the model, providing different points of views on how knowledge is manipulated, helps in the modelling. Such help can be obtained by modules that analyse the prototype. Examples of modules that can analyse the knowledge base according to the conceptual model and help in the instantiation of the model can be found in [Geldof & al. 94], [Gruber 89], [Paris & al. 93] or [Linster 93-b]. In the context of the elaboration of idiosyncratic models such modules must be constructed according to the model.

Note that prototyping as advocated here must not be confused with prototyping as intended in first generation expert systems, i.e. adding production rules to production rules without any abstract modelling [Lasque 86]. First, what is prototyped is the conceptual model of the system, and not the system itself ; the amount of knowledge tackled by prototyping is very different. Second, knowledge-level prototyping is based on high-level specific languages. Finally, this prototyping is used as a help to model, and not to implement the final system. We do not suggest a trial-and-error process, but advocate flexibility when modelling. We use the expression ‘knowledge-level prototyping’ to express the idea that the construction of the prototype aims at tackling modelling issues, and that the prototype can be studied at the knowledge-level.

Given this problematic, we describe in the next section the approach we propose to support our view of modelling.

3. The Mapcar approach

3.1. Principle

The final conceptual model is the result of a negotiation between the expert and the knowledge-engineer. A prototype reifying the model serves as a basis for this negotiation. Automatic modules analysing the prototype can be defined to allow different points of view on the model. Cycles composed of modelling and operationalisation phases are iterated until the obtaining of a satisfactory model:

• Modelling. The first version of the model is based on the expert’s knowledge (domain knowledge and problem-

solving examples). At the end of this phase the conceptual primitives (types of knowledge and high-level manipulations) and the structure of the model (the different levels, what is represented at every level and how levels interact) are identified. The definition of version i+1 is based on the critiques of version i, analysis of concrete problem-solving produced by the prototype that reifies version i, information on the model and the knowledge base provided by analysis modules and more expert knowledge. Any notions of version i (including the primitives) can be modified.

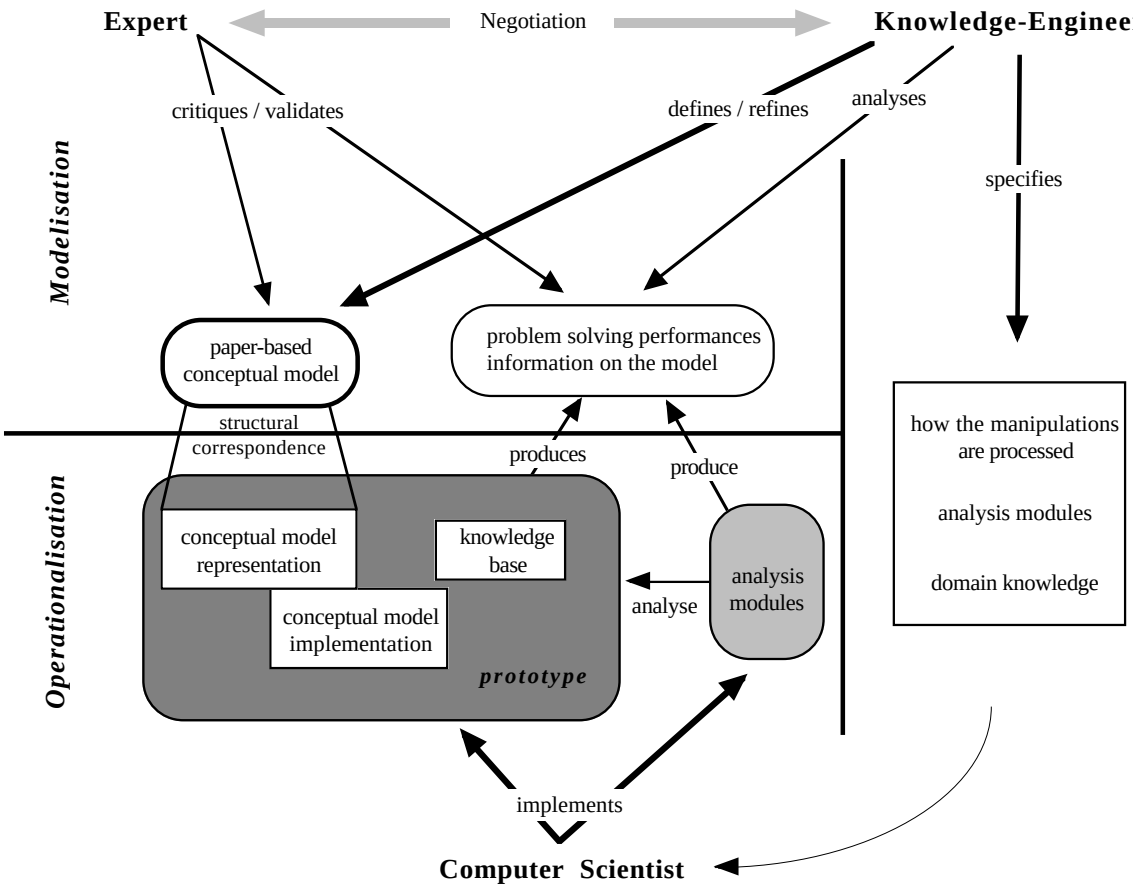
• Operationalisation. We distinguish two phases during the operationalisation. The representation of the model corresponds to the translation in the operationalisation language of the conceptual primitives and the high-level manipulations. At the end of this phase an unambiguous representation of the model is obtained and some automatic analysis (as matching pre/post conditions of some actions) is possible. The implementation requires the coding of the manipulations. At the end of this phase the operational prototype is completed and can be used to perform problem-solving. Note that during prototyping, the conceptual difference we make between representation and implementation does not necessarily correspond to two different phases ordered in the time.

During these cycles three (types of) persons interact: the expert, the knowledge-engineer and the computer scientist (Cf. Figure 3).

• The expert is the source of the knowledge and is responsible for the acceptance of the model.

• The knowledge-engineer is responsible for the modelling. He interacts with the expert to elaborate and refine the model and to instantiate it with the domain knowledge. He specifies and maintains a prototype that corresponds to the conceptual model with, if needed, the computer scientist. He specifies analysis modules if necessary.

• The computer scientist is responsible for the implementation of what is not directly provided by the framework.



The knowledge-engineer defines and then refines the conceptual model. The computer scientist implements and maintains a prototype and analysis modules from the specifications of the knowledge-engineer. The expert and the knowledge-engineer negotiate the model on the basis of the current paper-based version, and the problem-solving performances and information provided by the prototype and the analysis modules.

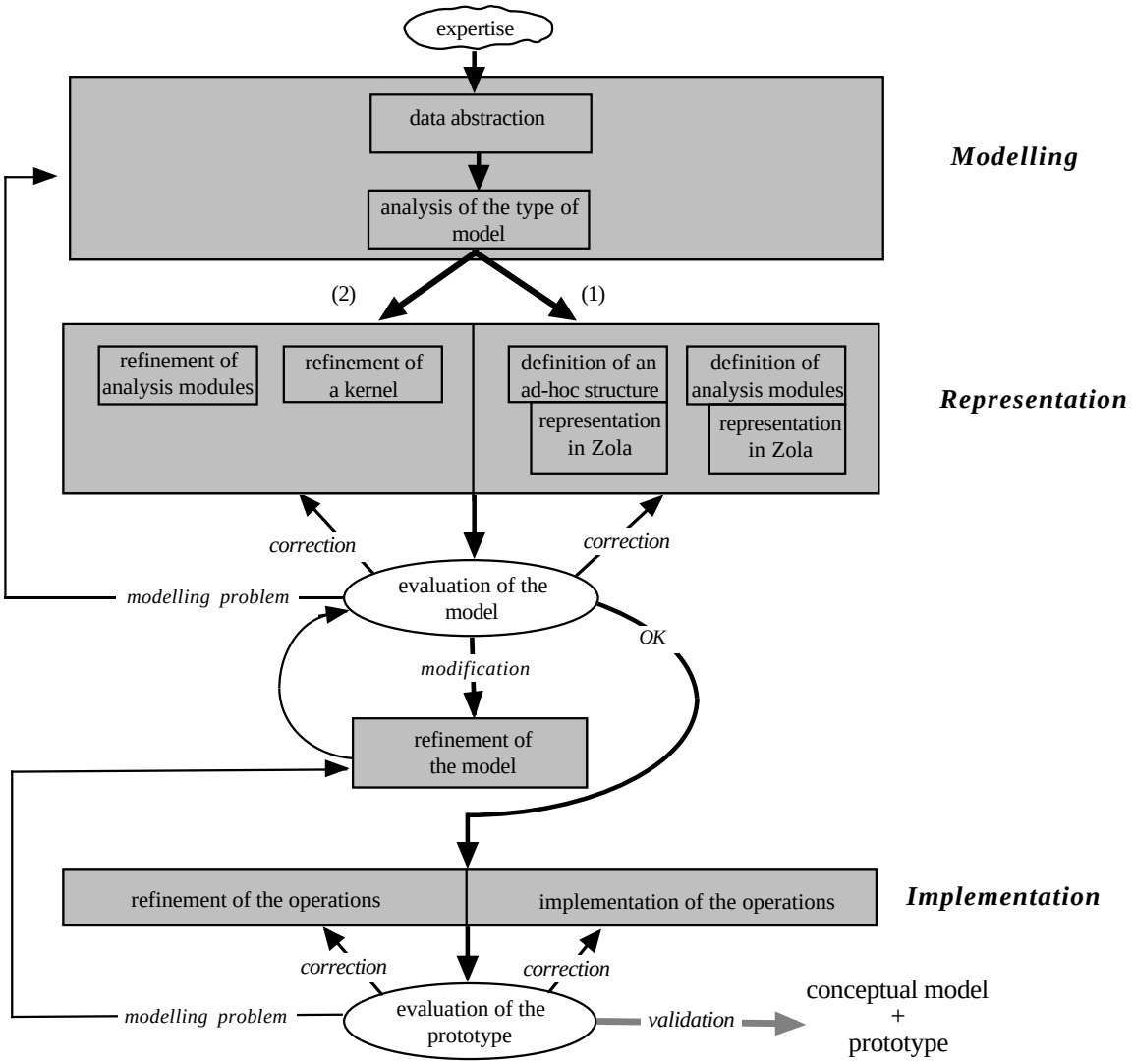
Figure 3. *The Mapcar approach.*

3.2. Elaboration of a conceptual model

We suggest two ways of elaborating the conceptual model (Cf. Figure 4):

(1) Elaboration of the model from scratch if the structure of the conceptual model that is being abstracted from the expertise is completely new. In this case, if analysis modules are necessary they must also be constructed from scratch according to the model.

(2) Reuse and refinement of a type of model if the structure of the conceptual model that is being abstracted from the expertise corresponds to an available structure.



From the expertise it appears that (1) an ad-hoc model must be constructed from scratch or (2) a predefined type of model can be refined. Once the model has been represented unambiguously, it is evaluated by the knowledge-engineer and the expert. This can conduct to correcting the representation if some errors occurred, a come back to the initial modelling, a refinement of the current model, or, if the model is validated, the implementation phase. This consists in coding the concrete manipulations, and allows the construction of an operational prototype. Evaluating what problem-solving is obtained and information provided by the analysis modules will conduct to correct the implementation, return to the refinement of the model or validate the current model. Note that (1) and (2) are not contradictory as (1) may be necessary to recognise (2).

Figure 4. *Elaboration of a conceptual model.*

We call 'type of model' a possible way to structure a conceptual model. A certain number of structures appear recurrently, such as for example:

- Chain of high-level actions: the problem-solving is modelled as a chain of high-level actions that each perform a

specific part of the problem-solving;

- Static decomposition of tasks: the problem-solving is modelled as a decomposition of tasks (that represent goals to achieve), each task being performed by a method;

- Dynamic selection of tasks and methods (DSTM): the problem-solving is modelled by defining tasks and methods, and a mechanism dynamically selects what task should be performed next and/or what method is the most pertinent to perform it.

Given a type of model, one can provide an operational skeleton where the type high-level notions are represented. As an example, the type of model DSTM provides structures to represent tasks and methods, a set of primitives such as select-a-task, identify-the-possible-methods, select-a-method (etc.) and a control over these primitives. To such a structure, analysis modules dedicated to this type of model can be associated. For example, for DSTM, one can define modules that check that every task can be solved by a method, that define implicit interferences of different methods, etc. Such a skeleton allows a quick operationalisation of models by:

- direct reuse, if the model to be operationalised corresponds to the kernel;

- instantiation, if the model to be operationalised only requires modification of some notions. One can modify the definition of a task and/or a method, how a primitive such as identify-the-possible-methods is achieved or the control over the primitives. Taking into account these modifications corresponds to the refinement of some identified parts of the kernel. Analysis modules must be modified in consequence, but remain pertinent;

- derivation, if the model to be operationalised requires new notions. In this case, what is reusable depends on the distance between the kernel notions and the model to operationalise.

From the point of view we adopt here, the structure of Kads models of expertise (strategy, task, inference, domain) [Wielinga & al. 92] is but one particular type of model, and a language such as Model-K is a kernel that corresponds to this type of model.

4. Putting into practice

This section describes the putting into practice of our approach. In subsection 4.1. we present a general description of the Zola operationalisation language. In the following subsections we present examples of how the key points of our approach can be put into practice with Zola: in 4.2. how one can represent and operationalise conceptual primitives, in 4.3 how one can adapt a type of model, in 4.4 how one can define analysis modules, in 4.5 what can be given to the expert in order to enable him to support the modelling process. Examples are taken from the Fiona project (elaboration of a conceptual model for fault diagnosis on robots [Tchounikine & al. 95, Istenes & al. 96-a]) and the DSTM type of model [Istenes & al. 96-b].

4.1. The Zola language

Zola proposes basic structures[7] that can be combined to define the conceptual primitives necessary for a particular model. Zola primitives are not to be used as conceptual primitives. Prototyping with Zola requires a step more than prototyping with a language such as Model-K: the elaboration of the conceptual primitives and the type of model that corresponds to the domain expertise one is modelling. This is the price to pay to allow the modelling of idiosyncratic structures, and, while refining the model, the capacity to make these primitives and/or the structure of the model evolve. Zola semantics are defined through its implementation.

The 4 Zola sublanguages

Zola proposes four sublanguages (Cf. Figure 5). A sublanguage enables the representation of some knowledge in a certain form and with certain characteristics. In accordance with our objective, a sublanguage does not correspond to a particular conceptual layer of a model, but aims to represent different materials necessary when representing a model.

[7] It can be argued that any structure can be considered as a conceptual primitive, and that a language cannot be neutral. We agree that a neutral modelling language has no sense. Zola is not a modelling language, its primitives are not to be used to directly model a behaviour.

• L3 for conceptual primitives. L3 allows the definition of conceptual primitives as types of knowledge. These types are represented as frames (sets of fields) that can be instantiated by the domain basic knowledge. Types can be structured in an inheritance hierarchy. A field can be instantiated by a literal, a list of literals, an instance of a knowledge type or a list of instances of a knowledge type. Types that require structured constructions can use predefined types as, for example, the 'collection' type.

• L2 for operational knowledge representation. L2 allows the definition of operations that manipulate knowledge represented with L3. The language proposes a set of predefined control structures (e.g. While and If-Then-Else) and L3 manipulation primitives (e.g. Match). Operations are defined by instantiating control structures with operations defined previously or manipulation primitives.

• L1 for static control. L1 allows the definition of algorithms over L2 operations. This allows the definition of the overall structure of the system high-level operations without having to use Zola primitives when this is not necessary. A criterion to use either L1 or L2 is that L2 structures have been defined in order to be analysable by reflexive modules, when L1 simply corresponds to Lisp[8].

• L4 for additional information. L4 allows the representation of additional information that can be attached to L2 and L3 objects. L4 is dissociated into four sublanguages in order not to mix information of different types. They all have the same power of expression as L3 (information is represented as frames).

- Technical information is divided into system information and operationalisation information. System information is information managed at an internal level (by the language interpreter) that can be of interest for modelling. An example is the CPU time of an operation, which can be used as a criterion for its use. Operationalisation information is information related to the implementation of the model that is not relevant at the knowledge-level, does not appear in the conceptual model, but has to be represented for operationalisation. An explicit representation of such notions can be necessary, for instance, for explanation purposes.

- Conceptual information is information that is not present (or not in an accessible way) in the representation of the knowledge types and operations, but is necessary to obtain a knowledge-level representation of a model. Conceptual information can be represented as conceptual handles, i.e. information attached to a knowledge type or an operation, or as a conceptual description, which is not attached to another particular component. As an example, 'what an operation does' or 'how it proceeds' are two forms of conceptual information that cannot easily be retrieved from the description of the operation in L2. They can be added to the operation with conceptual handles.

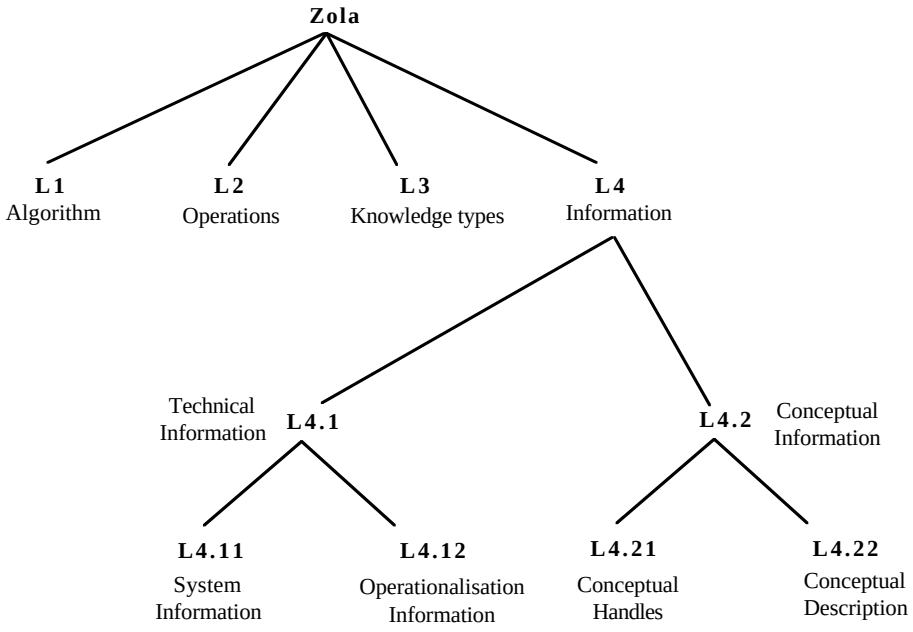


Figure 5. *The Zola four sublanguages.*

8 Zola is implemented in Lisp with an intensive use of an object-oriented Lisp-based dialect.

Using Zola to represent and operationalise a conceptual model

The representation of a conceptual model corresponds to:

(1) Definition of the knowledge types with L3 and, if necessary, of conceptual handles (L4.21) attached to them ;

(2) Definition of the high-level operations (i.e. operations that make sense at a knowledge-level) with L2 and, if necessary, of conceptual handles (L4.21) attached to them.

(3) Definition of the control structure. This can be achieved using L1 if the control structure is a simple algorithm on high-level actions, or it can be modelled as a specific KBS (see [Istenes 96-b] for an example of such a control);

(4) Definition of additional conceptual information about the model on its own (i.e. not related to specific components) with L4.22.

The implementation of a conceptual model corresponds to:

(1) Instantiation of the knowledge types by domain knowledge with L3;

(2) Definition of the technical information required for the operationalisation (L4.12);

(3) Definition with L2 of the intermediate and low-level operations that describe how to perform the high-level operations.

Operations defined with L2 can be dissociated according to their conceptual level: high-level operations are those used when representing the conceptual model. These high-level operations are then implemented by using intermediate operations that reference a knowledge type but not its fields, which themselves use low-level operations (that are completely domain dependent, i.e. directly manipulate the domain type fields).

Reflection in Zola

The classical definition of a reflective system is a system that can introspect itself. This initial definition is generally extended to architectures where a part of the system reasons on another part, or where a system (the reflective system) reasons on another system (the object system)[9]. In the context of our project we need two different types of reflection:

• Reflection to represent a particular aspect of a conceptual model.

This can be necessary to model strategic reasoning [Karbach & al. 93]. This type of reflection has been called "knowledge-level reflection" in [van Harmelen & al. 92]. The explicit representation of knowledge types and operations extended with additional conceptual information provides the necessary data. Knowledge-level reflection is thus possible through a direct access to the object system.

• Reflection to analyse how the model is implemented.

The construction of analysis modules can require structural reflection, i.e. the definition of reflective modules that can analyse how the model is implemented. For this purpose, the internal representation of L2 and L3 are isomorphic. Therefore, L2 operations can analyse and manipulate other L2 operations exactly as they can analyse and manipulate L3 knowledge types.

4.2. Operationalisation of a model (examples from DSTM type of model)

Tasks and methods are classical conceptual primitives that appear in different methodologies and are directly proposed by different operationalisation languages (as Lisa [Jacob-Delouis & al. 95], Tips [Punch & al. 93] or MML [Guerreiro-Rojo 95]). We present here the manner in which Zola enables the operationalisation of such primitives.

9 There is no general consensus about this definition of reflection that is called meta-reasoning in some other work.

Figure 6 shows how the knowledge type `method` and its two subtypes `decomposition-method` (a method that decomposes a task into subtasks) and `operational-method` (a method that solves a task) can be represented.

```
(type
       name : method
       subtype-of : L3
       slots :
              preconditions
              possible-results
              necessary-ressources
              favourable-context
)
```

```
(type
   name : decomposition-method
   subtype-of : method
   slots :
          decomposes-in
)
```

```
(type
name : operational-method
subtype-of : method
slots :
       performed-by
)
```

Figure 6. *Representation of knowledge types in L3.*

Figure 7 presents a high-level operation that identifies the possible methods that can perform a given task. It uses two intermediate operations to check the pertinent methods and the firable methods.

```
(operation
   name                 identify the possible methods
   profile              list of operations
   conceptual-handles
          operation-description : CH-identify-the-possible-methods
          operation-fails      explicit description of how the operation failure is defined
          operation-succeeds   explicit description of how the operation success is defined
   parameters
          in :   a-set-of-methods, a-task
          out :  list-pertinent-and-firable-methods
   list of operations :
          match a-set-of-methods list-pertinent-methods 'check-pertinent a-task
                 defines from a-set-of-methods pertinent methods according to the operation check-pertinent performed
                 with the parameter a-task
          match list-pertinent-methods list-pertinent-and-firable-methods 'check-firable a-task
)
```

Figure 7. *Manipulation of the type of knowledge 'method' in L2.*

To enable knowledge-level reflection such operations are described at a knowledge-level by different conceptual handles attached to the knowledge types and operations. Information such as rapidity or percentage of success are attached to the methods and can be considered when selecting a method. The description of the conceptual handle that describes the operation `identify the possible methods` is presented in Figure 8. This information can be used when a strategic layer defines which of the different possible operations that can achieve selection of methods will be used.

```
(instance         l4.21:conceptual-handles-that-describes-an-operation
   name           CH-identify-the-possible-methods
   I do           identify the possible methods
   I use          list-of-methods, current-task, selected-methods
   I succeed if   some possible methods exist
   I fail if      no possible methods are found
   Criteria       no user interaction
)
```

Figure 8. *Conceptual representation of an operation in L4.21.*

Although Zola does not directly propose conceptual primitives, one can see from these examples that such primitives can be easily defined and represented explicitly. When refining the model, eventual modifications of knowledge types

and/or operations can be undertaken by modifying identified parts of the prototype. In order to present comprehensible examples we have used classic structures; representing idiosyncratic knowledge types and knowledge manipulations can be processed identically.

4.3. Refinement of a type of model (examples from DSTM type of model)

Many KBS model, in some manner, the fact that at a given state of the problem-solving, the system defines what is the next task to achieve. Different works have pointed out that providing different possible methods to solve a task emphasises capacities such as flexibility or robustness (see [Benjamins 95] for example). Given a task, the system dynamically selects, at run-time, what method should be used. This selection can be achieved according to the current context of problem-solving and/or to some meta-level decisions (e.g. time constraints and user preferences). Different languages address the operationalisation of such a model as TroTelc [Vanwelkenhuysen & al. 90], Lisa [Jacob-Delouis & al. 95] Tips [Punch & al. 93] or MML [Guerrero-Rojo 95] (within this last reference can be found a comparison of some of the other languages).

DSTM (for Dynamic Selection of Tasks and Methods) is an operational kernel that allows a rapid operationalisation of conceptual models that fall into this type of model: the problem to solve is defined as a task; achieving a task can be performed by a method; a method can split a task into (sub)tasks (decomposition methods) or directly solve the task (action method); different methods can be used to solve a task.

Tasks and methods are represented using L3, high-level operations such as `identify the possible methods` are represented using L2 (Cf. examples presented in subsection 4.2). Two versions of the general strategy are available: a simple algorithm that iterates defining the task, the possible and then firable methods and achieve the selected one, and an explicit control that extensively uses knowledge-level reflection to select dynamically what operations and what criteria are used when performing the selection (see [Istenes 96-b] for further description).

This kernel can be instantiated by modifying task or method definitions. For instance one can modify a task's basic description to dissociate possible methods (the exhaustive list of methods that can achieve the task) and known methods (a non exhaustive list). This only requires the refinement of the L3 types and of how the high-level operation `identify the possible methods` is implemented to obtain a new behaviour:

(1) if possible methods have been defined for the task, the list is directly available.

(2) if known methods have been defined and some still remain unused, the list is directly available.

(3) if no method is directly defined as possible or the list of unused known methods is empty, a new list of methods is defined by comparing the expected results of the task with the results of the methods defined in the system.

An example of a derivation is the ZTM modelling language [Beaubeau & al. 96], which has been derived from DSTM in order to operationalise conceptual models constructed following Macao methodology [Aussenac & al. 94]. Macao models fall into the DSTM type of model. Some minor differences in the description of methods and tasks exist and would only require an instantiation. A major difference is that in Macao methods and tasks do not directly manipulate domain knowledge, but manipulate input and output roles that, for a given problem and at a given state of the problem-solving, are played by some domain knowledge (this is very similar to Kads notion of role). Another difference is the introduction of an algorithmic ordering of subtasks by decomposition methods.

Adapting our kernel to the operationalisation of Macao conceptual models therefore required the modification of knowledge types ('tasks' and 'methods') and of their manipulation operations, the definition of a new knowledge type ('role') and its taking into account. This implied modifying operations of the kernel, as, for example, the intermediate operation `check-pertinent` (see Figure 7) that now uses a new operation `interpret-roles`. The high-level operation `activate a method`, when it achieves a decomposition method, now attaches explicit constraints to the subtasks (represented as operationalisation information with L4.12) that represent their ordering. About 90% of the DSTM code has been reused.

4.4. Construction of analysis modules (examples from the Fiona project)

In the context of the Fiona[10] project we defined and operationalised a conceptual model of fault diagnosis for robots in plasturgy [Tchounikine & al. 95]. The aim was to define a high-level conceptual model that allows communication between the different people who have to cope with diagnosis problems: company customers, company trainers and customer technicians. This is one but an essential reason why it was decided to elaborate an idiosyncratic model that allows problem-solving corresponding to that of the company experts. This led us for example to model a 'cognitive-context' notion, which models the expert's cognitive framework at a particular state of its diagnosis process. During the diagnosis, human experts always consider they are "in a certain context" (for instance "there is no pertinent history, the gap is small, the robot has been restarted") and all their strategy and actions are linked to this context. Analysing what knowledge this context denotes emphasised that the model would be more rational if one represents differently this knowledge (for instance, in some cases, experts refer to a "small gap context" when it is in fact a large one !). But this would be a model different from the experts'.

The model that has been defined is based on different types of knowledge (e.g. `control knowledge`, `task realisation knowledge`, and `diagnosis knowledge`) that are manipulated according to their specificities by high-level operations (e.g. `definition of the task to be achieved`, `task performance`, and `definition of a diagnosis`). These high-level operations are iterated by the control algorithm presented in Figure 9 (Fiona's model is an example of the type of model 'chain of high-level actions').

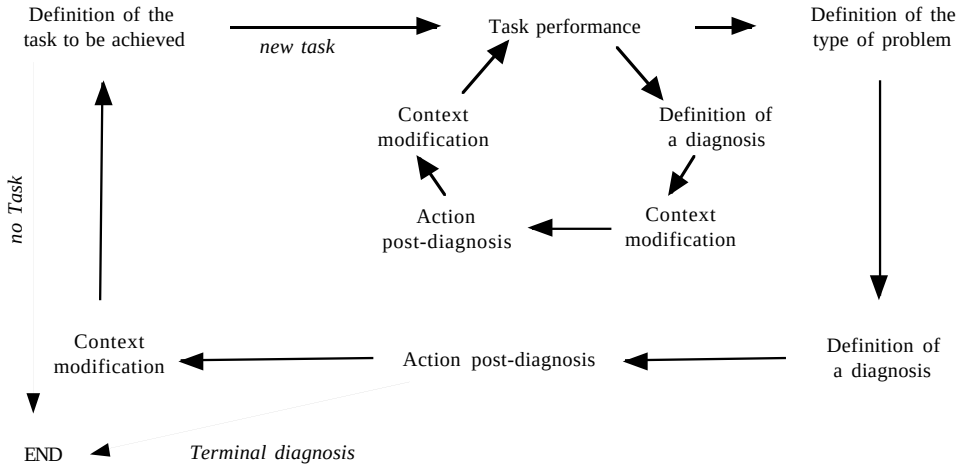


Figure 9. *Fiona's problem-solving strategy.*

In order to help in the modelling, we constructed modules that analyse the prototype and define additional information. These modules are specific to the model. They are implemented as L2 operations and, for the second example, use Zola's structural reflection capacities.

Analysis of particular aspects of the knowledge base

The different notions of the model are tightly linked. For instance to define the next task to be achieved one examines the current type of problem, the current cognitive-context, the states of the possible diagnosis, the tasks that have already been achieved and the basic data. In order to help in the control of how these notions interfere one with another, we constructed a module that defines all the cases where taking into account the cognitive-context notion leads to a difference in the strategy. This defines information as: *in this situation, with the cognitive-context C1, the task T1 will be considered, in the same situation, with the cognitive-context C2, the task T2 will be considered* (Cf. Figure 12). This analysis is based on scanning the knowledge base according to how the cognitive-context is manipulated by the operation `Definition of the task to be achieved`. Such a module is implemented using operations that manipulate the domain knowledge from an other point of view than the role they play in the problem-solving. This is possible because we are not limited to using predefined conceptual primitives.

10 Funded by the COMETT project 6617Cb and the "Pays de la Loire" (France) region.

Analysis of how the operations interfere with each other

In the Fiona model, a task corresponds to a concrete action to be achieved, typically a set of manipulations and interpretations. To every task corresponds some task realisation knowledge (procedures and expert knowledge) that represents how to achieve the task. For example, a task such as *qualify the failure* requires procedures such as *check the program*, *deduce the robot direction* and knowledge to define whether the robot went over the *initialisation point*, etc.

Task achievement is cognitive-context dependent. For this reason, while performing a task, one checks the eventual evolution of this context by scanning the 'diagnosis state' and the 'cognitive-context' knowledge (Cf. Figure 9). During the analysis of concrete problem-solving obtained with this model, it appeared that task performance is rarely modified by the cognitive-context evolution. To analyse more precisely this point of the model, we constructed a first module that defines, considering a particular high-level action, what slots of the corresponding type of knowledge are tested when examining if the action must be achieved (the condition-part of the action), and what slots are manipulated when the action is achieved (the action-part of the action). The intersection of the condition-part of the action *Task performance* and the action-part of the actions that check diagnostic states and the cognitive-context defines the set of notions that can evolve during the task performance and whose evolution can modify the performance of a task. A second module analyses the knowledge that performs a task, and determines if how the task is performed depends on some particular knowledge. Applying this to the knowledge that can evolve and modify task performance made different types of tasks apparent. Some do not depend on any of the notions that can evolve during the internal loop, some depend on these notions but are always performed with the same values for the corresponding slots, some vary upon these values. Interpreting these results by inspecting these different tasks appears relevant at the model level. Tasks from the first set are basic manipulations that depend on the robot technology. In contrast, tasks of the second and third set are bound to the point of view adopted in the modelling. And, for the tasks of the second set, the notions are referenced for information, just to note that this task has some sense in this context only.

This result indicates a possible refinement of the model. Although not apparent explicitly in the interviews of the expert, the dissociation of the different types of tasks is pertinent for a better understanding of the problem-solving process. This dissociation also allows the modification of the strategy in restraining the internal loop to tasks of the third set only. This is more cognitive, and allows better performances (on a classical case, the number of basic operations of the Zola interpreter went down from 26.000 to 22.000 with this optimisation).

4.5. What the expert is presented with

Our approach is based on the assumption that the domain expert can support the modelling, i.e. understand and analyse the model and the problem-solving it produces. We here present different examples of what the expert can be presented in order to allow his understanding.

Problem-solving (examples from the Fiona project)

Figure 10 and Figure 11 illustrate that maintaining the structural correspondence principle allows the presentation of the notions of the model while problem-solving, and an explicit representation of the domain knowledge.

```
...
[Current Task]           Quantify the gap
...
[Task Performance facts] the programmed point is 80 (which is near the initialisation point), the effective
                         point is 88, ...
[Cognitive-Context]      Small gap
...
[Diagnosis]              As the arm stopped before the programmed point a bad tuning of (...) is suspected
                         As the arm went over the initialisation point a coder failure is suspected
[Current Task]           Check tuning (this is a frequent cause of the current failure and can be easily
                         checked)
...
```

Figure 10. *A problem-solving episode.*

```
Name                R19
Type                control knowledge
Problem type        any problem
Cognitive-Context   no pertinent history, small gap, robot restarted
Actual diagnosis    coder failure suspected, bad tuning refuted
Basic data          stop before the programmed point, arm over the initialisation point
New task            change coder
```

Figure 11. *Control knowledge in Fiona.*

Different points of view on the domain knowledge (examples from the Fiona project)

Figure 12 presents examples of information provided by analysis modules.

```
In the cognitive-context ...
The task Qualify-failure
Is achieved after the task Quantify-failure
as it produces the information: ....
that is necessary to achieve Quantify-failure

In the situation :
new failure, test intermittent failure
in the cognitive-context : restart
The task: Definition-info-FVLZ is selected
In the same situation
in the cognitive-context : no pertinent history, restart, small gap
The task: Quantify-failure is selected
(justifications: ...)
```

Figure 12. *Different points of views on the domain knowledge.*

Structured explanations (examples from DSTM type of model)

The DSTM type of model is complex. In order to allow a synthetic understanding of how its different notions interfere during problem-solving, we have defined explanation modules. The objective is to present in concrete terms informations related to a particular notion. For example, the explanation module `Focus on a method` presents information on a particular method, e.g. its links with the tasks and comparisons with the other methods that can perform the same task (Cf. Figure 13). What is presented is constructed following an explanation strategy that addresses explicit objectives [Trichet & al. 96]: `Focus on a method` aims at the understanding of what implications the selection of a method can have on the rest of the problem-solving. The presented information, which is not limited to the one that can be used in one particular problem-solving, is structured according to this objective. The strategy is modelled and implemented with Zola as a reflexive module.

```
Link  task/method
   the method M1 can achieve tasks T1 and T4
         M1 is defined as possible for T1
         M1 is defined as known for T4
   ...
Competitive  methods
   for T1
         T1 can also be achieved by M2
         M2 produces the results ...
   ...
Analysis  of  the  competitive  methods
   for T1
         M1 and M2 have the same preconditions but different favourable contexts
               in the context ... T1 will be achieved by M1
               in the context ... T1 will be achieved by M2
 ...
Execution  of  the  competitive  methods
   after M1 has been performed tasks T3 and T5 can become candidate
         M1 produces ... that is corresponds to T3 prerequisites
   ...
```

Figure 13. *Explanation 'Focus on a method' (episode).*

5. Discussion

5.1. On knowledge-level prototyping

We have presented how we use knowledge-level prototyping to elaborate idiosyncratic models. This approach can be used for other purposes such as:

• to help select a generic model from a library. Top-down methodologies advocate the reuse of generic models, but their putting into practice nevertheless requires an abstraction step. KLP can be used to support this step, i.e. to construct a first version of the conceptual model (what is called a "framework of the model" in Macao) to structure rough data and serve as a basis for the selection of a generic model.

• to achieve small size projects or to evaluate large projects. As an example, in the Fiona project, although knowledge based systems appeared *a priori* as appropriate tools, the company did not want to engage itself in a large scale project without an understanding of how the system would look, and some evidence that it was feasible. In other terms, what the company wanted was both a conceptual model as a means to understand the problem-solving expertise and how it can be modelled in a knowledge based system, and a running prototype to be sure that the abstract specification can be converted into a concrete software that fits their needs. Knowledge-level prototyping appeared as a pertinent approach for such a situation.

• to refine paper-based conceptual models defined using a general methodology. Model-K [Karbach & al. 93] and Omos [Linster 93-a, Linster 93-b] languages have been constructed for this purpose. Linster in particular advocates operationalisation by continuous refinement, claiming that new or underestimated problems only appear when a concrete experimentation takes place.

5.2. Zola and knowledge-level prototyping

The adequacy of a language for knowledge-level prototyping must be judged according to the objective of constructing, representing and testing the model. The fact that it does not allow the representation of the domain knowledge in the most pertinent way is not critical as far as it permits the representation of some knowledge for testing purposes (when constructing the model, one does not need to acquire all the domain knowledge, but only a representative subset that will allow its testing). In the same way, the fact that the language can also be used to implement the final system is not pertinent.

Zola is but one possible language that allows the putting into practice of our approach. Its features that are particularly useful are structures to construct model-specific types of knowledge, primitives to construct model-specific actions and structures to add conceptual information to types of knowledge and actions; the capacity to respect the structural correspondence principle and to capture the conceptual model in the operational prototype; the knowledge-level and structural reflection capacities.

5.3. Zola and modelling languages

Most operationalisation languages developed by the knowledge acquisition community are modelling languages: they force to a particular type of model. The fact that Zola objectives are different renders a comparison difficult. Zola emphasises flexibility while modelling, when modelling languages emphasise reusability and genericity based on a particular point of view of what is a conceptual model. Zola addresses both knowledge-level reflection (for modelling meta-level control activities) and structural reflection (for constructing analysis modules), when reflexive modelling languages (such as Model-K or Lisa) are not concerned by constructing analysis modules. Zola addresses prototyping the different components of the model, when modelling languages designed for prototyping (such as Model-K) don't address prototyping the conceptual primitives.

From the perspective of what has to be coded in order to operationalise a model, Zola is not competitive. With Zola one has to define the conceptual primitives we need and make them operational first (an exception is if the model

corresponds exactly to a predefined type of model such as DSTM). Moreover, Zola does not emphasise modelling domain knowledge. If the domain knowledge that is necessary to prototype the model cannot be represented using L3, we must use (such as MoMo [Fensel & al. 94]) an other language reachable from our Lisp environment.

From the perspective of what aspects of the model can be lost while operationalising, Zola is competitive for models that don't gently slip into a predefined structure. For such models, Zola can be viewed as a language that permits constructing an adapted modelling language (ZTM for Macao models [Beaubeau & al. 96] is a good example). Constructing such a modelling language is easier in Zola than with a low-level implementation language.

What Zola has been constructed for, i.e. supporting the elaboration of models by data-abstraction using knowledge-level prototyping and reflexive analysis tools, is not addressed by modelling languages.

5.4. Zola and reflection

Our view on reflection is slightly different from the one that has been suggested by the Reflect project [van Harmelen & al. 92]. In Reflect, reflection is used for problem-solving, not for analysing the implementation. Reflect only addresses knowledge-level reflection, and suggests that instead of a direct access to the object system structures (the implementation), one should construct reflective systems that reason on a conceptual model of the object system. In this separate layer, dedicated to the objectives of the reflective system, aspects of the conceptual model that are not any more present in its implementation and/or cannot be retrieved from it can be represented. In our case, we need both knowledge-level and structural reflection. To keep the language homogeneous, we choose to add conceptual information to the object system rather than to undertake a separate description, which aims at objectives we do not consider (adding a reflective-layer to a preexisting system), and requires a causal connection between the object system and its abstract representation.

5.5. On the elaboration of the model in an educational context

There is no claim that using KLP to elaborate idiosyncratic models is always the optimal approach and that refining predefined models is never the good solution. For the objective we consider the two approaches have opposite strengths and weaknesses. An advantage of refining predefined models in the context of an educational software is explained in [Winkels & al. 92]. When teachers are not able to propose a satisfactory model for the studied domain, refining a generic model helps them in defining it. We think that such an approach induces problems that can make it not suitable, in particular the definition of a model that solves problems but is not pedagogically suitable and/or that implies modifying all the pedagogy around the software, which is not necessarily possible. The intrinsic weakness of the approach we suggest is to fail in the abstraction process. This will conduct to the definition of a descriptive model of the teacher apparent behaviour, when one should define a prescriptive model of how students should solve problems. Our assumption is that teachers involved in the construction of the KBS are able to define what should be presented by the educational software. A certain number of experiences [Choquet & al. 95, Tchounikine 94] suggest this is a reasonable assumption.

Acknowledgements

The Zola language has been implemented by Z.Istenes (IRIN) as part of his thesis work. The Fiona project has been achieved with C.Choquet (IRIN) and Z.Istenes, DSTM with F.Trichet (IRIN) and Z.Istenes, ZTM with D.Beaubeau (IRIN) and N.Aussenac-Gilles (IRIT, Toulouse, France). Thanks to Angi Voss (GMD, Sankt Augustin, Germany) and Philippe Laublet (Laforia, Paris, France) for their comments on an earlier version of this paper.

References

[AI-Watch 95] *AI-Watch special issue on the EuroKnowledge project*. (1995) **4**(8).

[Anderson & al. 90] Anderson J.R., Boyle C.F., Corbett A.T., Lewis M.W. (1990). Cognitive modelling and intelligent tutoring. *Artificial Intelligence* **42**:7-49.

[Aussenac 94] Aussenac-Gilles N. (1994). How to combine data abstraction and model refinement: a methodological contribution in MACAO. In (Luc Steels, Guus Schreiber and Walter Van de Velde editors) *Lecture Notes in Artificial Intelligence (867) 'A future for Knowledge Acquisition':*262-282. Springer-Verlag, Berlin, Germany.

[Aussenac & al. 94] Aussenac-Gilles N., Matta N. (1994). Making the method of problem solving explicit with Macao. *International Journal on Human-Computer Studies* **40**:193-219.

[Beaubeau & al. 96] Beaubeau D., Aussenac-Gilles N., Tchounikine P. (1996). Mona au pays des rôles: opérationalisation de modèles conceptuels Mona en Zola. *IRIT internal report IRIT/96-23-R*, Toulouse, France.

[Benjamins 95] Benjamins R. (1995). Problem-solving methods for diagnosis and their role in knowledge acquisition. *International Journal of Expert Systems: Research and Applications* **8**(2):93-120.

[Breuker 93] Breuker J. (1993). Reusable components for artificial problem solvers: the common Kads library experience. In the *Proceedings of the IJCAI workshop on knowledge sharing and information exchange*. Paris, France.

[Budde & al. 91] Budde R., Kautz K., Kuhlenkamp K., Züllighoven H. (1991). *Prototyping. An approach to evolutionary system development*. Springer-Verlag, Berlin, Germany.

[Chandrasekaran 87] Chandrasekaran B. (1987). Generic tasks in knowledge based reasoning: high-level building blocks for expert system design. In *Proceedings of the 10th IJCAI*:1183-1192. Milano, Italy.

[Choquet & al. 95] Choquet C., Tchounikine P., Trichet F. (1995). Exploitations pédagogiques d'un modèle de raisonnement : une expérience en milieu industriel (in French). *Environnements Intéractifs d'Apprentissage avec Ordinateurs*, Tome 2:101-112. Eyrolles, Paris, France.

[Clancey 87] Clancey W.J. (1987). *Knowledge based tutoring. The Guidon program*. The MIT press, Cambridge, Massassuchets, USA.

[Elliot & al. 95] Elliot M.J., Bull H.T., Pulford C.I., Shadbolt N.R., Smith W. (1995). Constructive knowledge engineering. *Knowledge Based Systems* **8** (5):259-267.

[Fensel & al. 94] Fensel D., van Harmelen F. (1994). A Comparison of Languages which Operationalise and Formalise KADS Models of Expertise. *The Knowledge Engineering Review* **9**:105-146.

[Frederiksen & al. 93] Frederiksen J.R., White B.Y. (1993). The avionics job-family tutor: an approach to developing generic cognitive skills within a job-situated context. In the *Proceedings of the International conference on Artificial Intelligence in Education* (AI-ED'93):513-520. Edinburgh, Scotland.

[Gaines & al. 93] Gaines B.R., Shaw M.L.G., Woodward J.B. (1993). Modelling as framework for knowledge acquisition methodologies and tools. *International Journal of Intelligent Systems* **8**:155-168.

[Geldof & al. 94] Geldof S., Slodzian A. (1994). From Verification to Modelling Guidelines. In (Luc Steels, Guus Schreiber and Walter Van de Velde editors) *Lecture Notes in Artificial Intelligence (867) 'A future for Knowledge Acquisition':*226-243. Springer-Verlag, Berlin, Germany.

[Gruber 89] Gruber T. (1989). A method for acquiring strategic knowledge. *Knowledge Acquisition* **1**:255-277.

[Guerrero-Rojo 95] Guerrero-Rojo V. (1995). MML, a modelling language with dynamic selection of methods. In the *Proceedings of the Knowledge Acquisition for Knowledge-Based Systems Workshop* (Banff'95). Banff, Canada.

[Istenes & al. 96-a] Istenes Z., Tchounikine P. (1996). Zola: a language to Operationalise Conceptual Models of Reasoning. *Journal of computing and information* (special issue ICCI'96) **2**(1):689-706.

[Istenes & al. 96-b] Istenes Z., Tchounikine P., Trichet F. (1996). Dynamic Selection of Tasks and Methods as a Knowledge Level Reflective activity. In (A.Ramsay editor) *Frontiers in Artificial Intelligence and Applications, Artificial Intelligence: Methodology, Systems, Applications* (Proceedings of the 7th International Conference AIMSA'96):168-177, IOS Press.

[Jacob-Delouis & al. 95] Jacob-Delouis I., Krivine J.P. (1995). LISA: un langage réflexif pour opérationaliser les modèles d'expertise (in french). *Revue d'Intelligence Artificielle* **9**(1):53-88.

[Karbach & al. 93] Karbach W., Voss A. (1993). Model-K for prototyping and strategic reasoning at the Knowledge Level. In (J.P.David, J.P.Krivine and R.Simmons editors) *Second Generation Expert Systems*:721-745. Springer-Verlag, Berlin, Germany.

[Kieras 88] Kieras D.E. (1988). What mental model should be taught : choosing instructional content for complex engineered systems. In (J.Psotka, L.D. Massey and S.A.Mutter editors) *ITS: lessons learned*:85-111. Lawrence Erlbaum Associates Publishers.

[Laske 86] Laske O. (1983). On competence and performance notions in expert system design: a critique of rapid prototyping. In the *Proceedings of Les systèmes experts et leurs applications, journées internationales d'Avignon* (Avignon'83):257-297. Avignon, France.

[Linster 93-a] Linster M. (1993). Closing the gap between modelling to make sense and modelling to implement systems. *International Journal of Intelligent Systems* **8**:209-230.

[Linster 93-b] Linster M. (1993). Integrating conceptual and operational modelling: a case study. *Knowledge Acquisition* **5**:143-171.

[Means & al. 88] Means B., Gott S.P. (1988). Cognitive task analysis as a basis for tutor development: articulating abstract knowledge representations. In (J.Psotka, L.D. Massey and S.A.Mutter editors) *ITS: lessons learned*:35-57. Lawrence Erlbaum Associates Publishers.

[Newell 82] Newell A. (1982). The knowledge level. *Artificial Intelligence* **18**:87-127.

[Paris & al. 93] Paris C., Gil Y. (1993). EXPECT: intelligent support for knowledge base refinement. In the *Proceedings of the European Knowledge Acquisition Workshop*:220-236. Springer-Verlag, Berlin, Germany.

[Punch & al. 93] Punch W.F., Chandrasekaran B. (1993). An Investigation of the Roles of Problem-Solving Methods in Diagnosis. In (J.P.David, J.P.Krivine and R.Simmons editors) *Second Generation Expert Systems*:673-688. Springer-Verlag, Berlin, Germany.

[Reinders & al. 91] Reinders M., Vinkhuyzen E., Voss A., Akkermans H., Balder J., Bartsch-Spörl B., Bredeweg B., Drouven U., van Harmelen F., Karbach W., Karsen Z., Schreiber G., Wielinga B. (1991). A conceptual modelling framework for knowledge-level Reflection. *AI Communications* **4** (2/3):74-87.

[Sandahl 94] Sandahl K. (1994). Transferring knowledge from active expert to end-user environment. *Knowledge Acquisition* **6**:1-22.

[Schaafstal & al. 93] Schaafstal A., Schraagen J.M. (1993). Training of systematic diagnosis: a case study in electronic troubleshooting. In the *Proceedings of the International conference on Artificial Intelligence in Education*:529-535. Edinburgh, Scotland.

[Steels 90] Steels L. (1990). Components of expertise. *AI Magazine* **11**(2):29-49.

[Tchounikine & al. 95] Tchounikine P., Choquet C. (1995). Fault diagnosis expert system for robots: a knowledge level prototyping experience. In the *Proceedings of the Seventh International Conference on Software Engineering and Knowledge engineering* (SEKE'95):268-274. Rockville-Washington, USA.

[Tchounikine 94] Tchounikine P. (1994). Elaboration of a model of reasoning by prototyping at the Knowledge Level (in French). *Revue Sciences et Techniques Éducatives* **1**(4):483-501.

[Tchounikine 90] Tchounikine P. (1990). Representing and activating knowledge in educational software for a human instructor-like reasoning-process justification. In the *Proceedings of the International Conference on Advanced Research on Computers in Education* (ARCE'90):129-134, IFIP, Tokyo, Japan.

[Trichet & al. 96] Trichet F., Tchounikine P. (1996). Des explications pour le concepteur d'un système à base de connaissances (in French). In the *Proceedings of the French meeting on explanations* (Journées Explication'96):214-229. Sophia-Antipolis, France.

[Valente & al. 93] Valente A., Breuker J., Bredeweg B. (1993). Integrating modeling approaches in the CommonKADS library. In (A.Sloman, D.Hogg, G.Humphreys, A.Ramsay and D.Partridge editors) *Prospects for Artificial Intelligence* (Proceedings of the AISB'93):121-130. IOS Press, Amsterdam, The Nederlands.

[van Harmelen & al. 92] van Harmelen F., Wielinga B., Bredeweg B., Schreiber G., Karbach W., Reinders M., Voß A., Akkermans H., Bartsch-Spörl B., Vinkhuyzen E. (1992). Knowledge-Level Reflection. In (B. Le Pape and L. Steel editors) *Enhancing the Knowledge Engineering Process, Contributions from ESPRIT*:175-204. Elsevier Science, Amsterdam, The Netherlands.

[Vanwelkenhuysen & al. 90] Vanwelkenhuysen J., Rademakers P. (1990). Mapping a Knowledge Level analysis onto a computational framework. In the *Proceedings of the European Conference on Artificial Intelligence* (ECAI'90):661-666.

[Wenger 87] Wenger E. (1987). *Artificial intelligence and tutoring systems, computational and cognitive approaches of the communication of knowledge*. Morgan Kaufman Publishers inc.

[Wielinga & al. 92] Wielinga B, Schreiber A., Breuker A. (1992). KADS: a modelling approach to knowledge engineering. *Knowledge Acquisition journal* **4**(1):1-162.

[Winkels & al. 92] Winkels R., Breuker J. (1992). Modelling expertise for educational purposes. In the *Proceedings of the International Conference on Tutoring Systems* (ITS'92):633-641. Montréal, Canada.